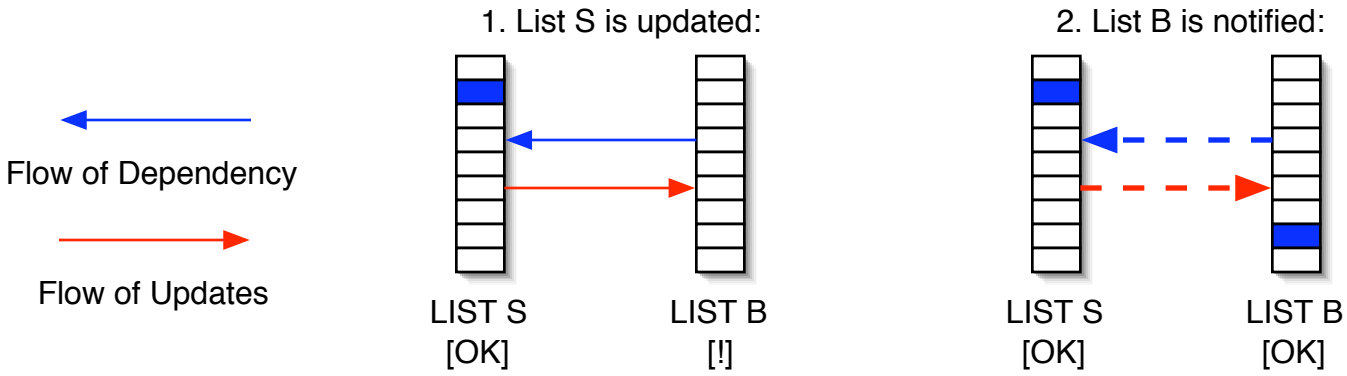
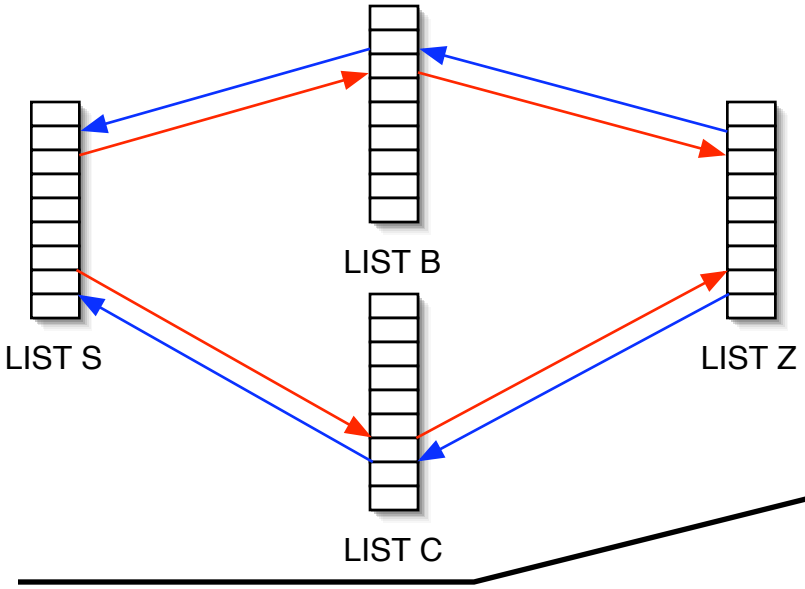
Glazed Lists: Managing Notification Dependencies with ListEventPublisher

January 2005
http://publicobject.com/glazedlists/

Jesse Wilson, Lead Developer
O'Dell Engineering Ltd.

After an EventList is updated, it updates all listening EventLists. Dependent EventLists are in an undefined state after their source is updated, but before they have received notification.

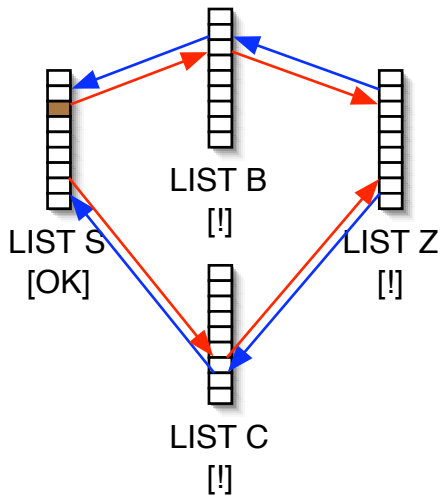




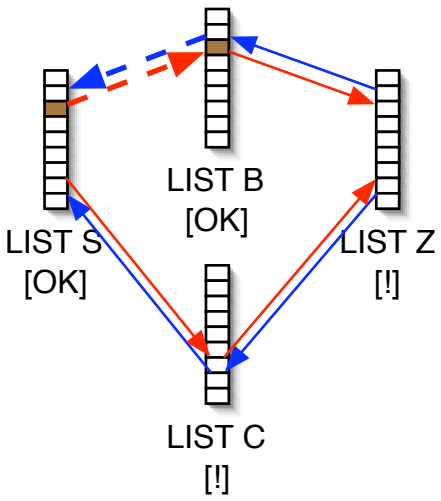
Suppose we have an EventList, Z, that depends on two EventLists, B and C. Each of these EventLists depends on the source EventList, S.

The EventList Z can depend on multiple source EventLists because it may filter one EventList based on the contents of the other.

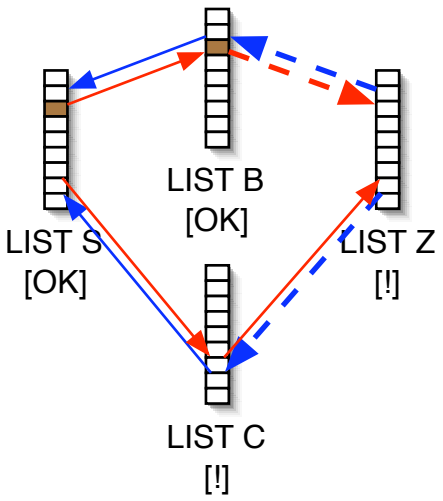
Suppose S is modified.



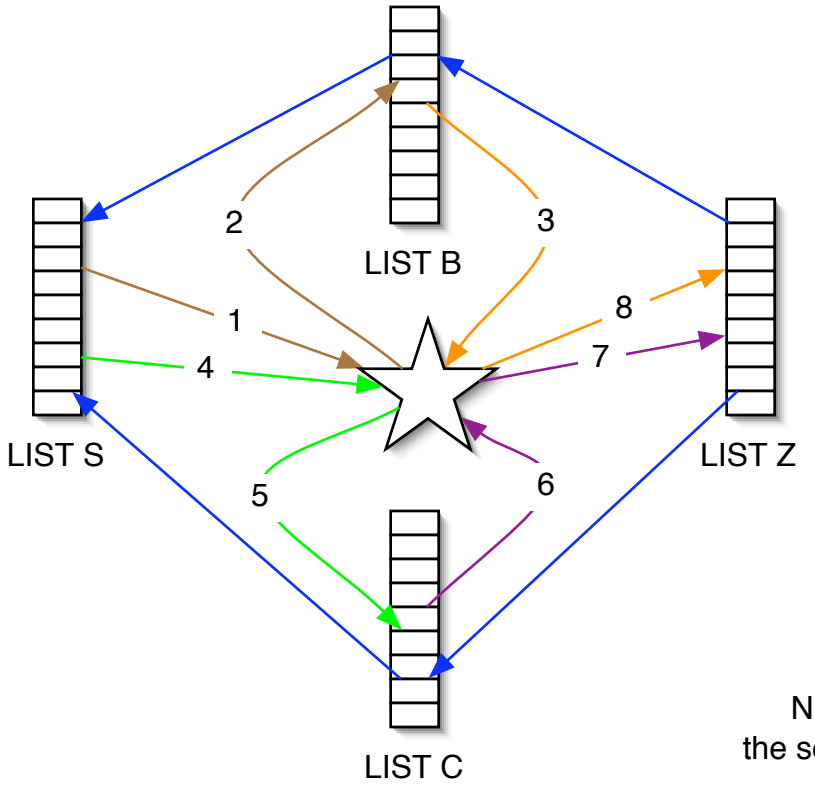
Then S will update B.

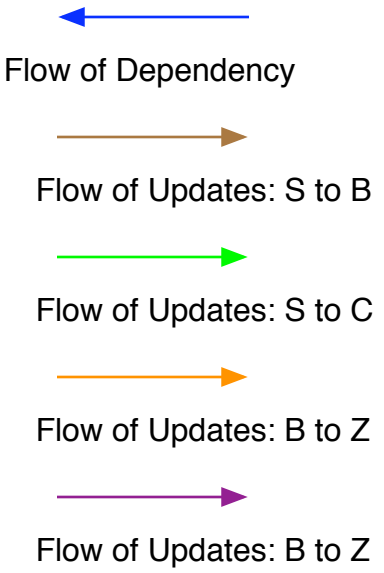


Then B will update Z. But Z depends on C, which is now undefined because it has yet to be notified by S.



So we need to change the notification order from { S, B, Z, C } to { S, B , C, Z }. Although in this case the result is breadth-first, in general the only safe notification order may be arbitrary. Therefore the ListEventPublisher manages all update notifications.





Numbers on flow liness indicate the sequence in which they are called.

The ListEventPublisher acts as a service for notifying listeners of change events with proper dependency management. All connected EventLists share one ListEventPublisher so that it has a global view of notification dependencies.

When an EventList tells the ListEventPublisher to fire an update event to a listener, the update event may be queued so that the listener's other dependencies may be brought up-to-date as well. To accomplish this, the ListEventPublisher maintains a queue of unfired events. Because one instance is shared, the method firing events has access to the same queue, which can be updated and as necessary.

The ListEventPublisher's fireEvent() method is reentrant, making it possible to pass queues of unsatisfied listeners up and down the call stack.

Note that this problem is orthogonal to concurrency and locking. The notification sequence problem exists in both multi- and single-threaded environments. The ListEventPublisher class is not thread-safe but may be used concurrently with an external locking strategy.

Note that sometimes the ListEventPublisher cannot automatically infer a dependency and it must be configured explicitly. For example, a dependency may also exist via a GUI listener such as a Swing ListSelectionListener or an SWT SelectionListener. To accomplish this, use EventList.getPublisher().addDependency(EventList, ListEventListener).